



NEW HAMPSHIRE 99'ers

MAY 1987
NEWSLETTER VOL 5 NO. 5
POB 5991 MANCHESTER, NH 03108

>OLD

The last meeting saw another totally screwed up demonstration. Yours truly proceeded to show everyone just how much he didn't know about LOGO. Fortunately, other members jumped in and provided some relief. Fear not! LOGO shall rise again!

Welcome back, Richard Bailey! Richard is our club librarian and puts countless hours into keeping our library in excellent shape. A listing of disks is enclosed in this issue.

Chris Agrafiotis had very favorable comments on Mark Beck's Creative Filing System. Chris is a recognized expert on data bases - not necessarily how to write them, but definately how to use them. Chris probably has purchased every data base program written for the TI and has found the majority of them lacking. His initial comments about Beck's CFS were refreshingly upbeat (instead of beat up).

A slate of officer candidates was nominated from the floor and is as follows:

- President Paul Bendeck
- Vice President Chris Agrafiotis
- Treasurer Bob Bouchard
- Secretary Curtis Alan Provance

I'm quite pleased with the proposed new officer slate. I think you'll find Paul and Chris a very effective leadership team. Thanks also to Bob for taking over the club's immeasurable wealth.

Don't forget, however, that this is not a Soviet style election. You may vote for whomever you please. The above candidates have the edge in that they are willing to serve. Thank you in advance! Elections will be in June.

The Treasurer reports that we have over \$600 in the bank, with some minor bills to take care of. I requested that the officers be allowed to scout out the best deal on a color monitor - no one objected. With the acquisition of same, our club system will be complete.

>NEW

Next month, Paul Bendeck will give a proper LOGO demonstration. Paul has probably forgotten more about LOGO than I ever knew. He has written a game (and other programs?) which he will demo for us.

In July, Chris Agrafiotis will give us the complete story on Mark Beck's Creative Filing System. Chris knows how to tear apart a data base, so this should be interesting.

As I close, let me mention the nicest 'new' thing in the club - members. In the last two months we have picked up several members. The TI is not dead - or even sleeping! If you know someone who owns a system ('for my kids...') drag them along to a meeting or share your newsletter with them. I also encourage you to ask questions about BASIC, XBASIC, TI-WRITER, or whatever. If I can't answer them, I'll find someone who can.

The C Column

=====

Jim Jagielski
Route 2, Box 626
Sanbornville, NH 03872
(603) 522-8952

This month we'll briefly go though the process of compiling and running a c99 program and, in addition, I want to elaborate on inline push code. If you don't know what inline push code is please be patient. We'll get to it very soon.

First thing first, when typing in a c99 program with TI-WRITER make sure it's saved with the PF option. (Ed. note - this prevents the TAB settings from being appended to the end of the file) With that in mind, carefully type in Demo#1. Demo#1 is a graphic program which attempts to exploit some of c99's speed and elegance. Due to the lack of a better name I've labeled the program "Bouncing Ball." Demo#1 uses graphic functions that are similar in appearance to some BASIC commands. I don't recommend equating BASIC with C, but for this particular Demo see if you can implement a BASIC version of it. I hope you like this graphic demo because subsequent ones will not be graphic. Future demos will be straight forward and to the point, hence not as pretty.

Okay, do you have the first demo ready to compile? Load the c99 compiler (version 2.1). Find the disk(s) with the files GRFIRF and RANDOM;C and place it in disk drive#1 and the output disk in a different drive. If you only have one disk drive copy GRFIRF and RANDOM;C onto the output disk that you're using. These two files must be in drive#1 because during compilation the compiler will look for them in drive#1 (NOTE: the drive number which contains GRFIRF and RANDOM;C can be changed by altering the "#include" name in the Demo source code). The first prompt will ask if you would like to include text. Entering <y> informs the compiler to include the c99 source code as comments in the assembler source code. Unless you have a good reason, I don't recommend selecting this option. Selecting this option may provide you with an unnecessary long assembler source file. For this demo press <enter> or <n>.

The next option is the inline push code. Let's take a time-out from compiling the demo and investigate inline push code. If you were to look at an assembler source code produced by the the c99 compiler, you may find several B 15 assembler instructions at various location in the source code. B 15 forces a branch to the first address of the push routine (located in the C support object file - CSUP) which "pushes" a value onto the c99 stack. Stacks are the subject of another column, but it would suffice to say that a stack is a dynamic buffer - dynamic in that the buffer

grows when values are pushed onto the stack and shrinks when values are popped off the stack. c99 uses this data structure to store variables (i.e. local variables), values, and return addresses. The push routine is comprised of two primary instructions:

```
DECT 14
MOV 8,
```

The explanation of these two instructions will be addressed in a later column when I cover stacks and the c99 environment. Essentially, by selecting the inline push code option, every B 15 instruction in the assembler source code is replaced with a DECT 14 and MOV 8,. To the assembly language programmer this resembles a macro, because there seems to be a visible direct text substitution. As you can probably deduce, if inline push code is selected, the assembler source code will be slightly longer than normal, therefore the program will take up more memory. Why use this option? The answer is speed: the B 15 instruction requires several clock cycles to complete a branch and return. By have the push routine directly embedded in the code there is no need to branch and return, hence we save some time.

A good habit is to ask yourself two questions about your C programs. (1) Is memory space a big factor in this program? (2) Does this program really need to execute as fast as possible? If you answered yes to question #1 then you probably wouldn't want to select inline push code option. On the other hand, if you answered no to question #1 and yes to question #2, then you probably should select the option. I've written quite a few c99 programs for myself and about 80% of the time I don't select the option. It all depends on the type of program and its application.

How much speed is really gained with inline push code? To answer this question I wrote the program labeled Demo#2. First I ran a version of the program with NO inline push code for ten trial runs while concurrently observing the execution times. I then did the same thing with an inline push code version. The results of my experiment follow:

```
trial# | NO, inline | YES, inline |
-----+-----+-----+
 1 | 10.36 sec. | 10.14 sec. |
 2 | 10.32 sec. | 10.05 sec. |
 3 | 10.26 sec. | 10.01 sec. |
 4 | 10.28 sec. | 10.00 sec. |
 5 | 10.25 sec. | 10.04 sec. |
 6 | 10.30 sec. | 10.00 sec. |
 7 | 10.37 sec. | 10.01 sec. |
 8 | 10.27 sec. | 10.04 sec. |
 9 | 10.29 sec. | 10.06 sec. |
10 | 10.30 sec. | 10.05 sec. |
-----+-----+-----+
average: 10.30 " vs. 10.04 "
```

* execution time difference of 0.26 seconds or 2.52%

With more branches (i.e B 15) the time difference would be more apparent than it is now. Is the 2.52% execution time decrease significant? Well, I've presented the facts. I'll leave it up to you to answer that question for yourself. When programming in C always keep in mind the time and memory tradeoffs. The tradeoffs should be evaluated for each individual program.

Now, let's get back to compiling Demo#1. For the inline push code option enter a <y> for YES, because Demo#1 doesn't hog a lot of memory and time is somewhat of a critical factor. The next prompt is the name of the c99 source code to compile. Enter the filename which you gave Demo#1. The next prompt (i.e. output prompt) is the assembly source code filename. Enter an appropriate output filename. This output file is going to be assembled by the TI-Assembler. If any error messages are received during compilation, exit the compiler and examine your Demo#1 source file for typos and recompile.

At this point load the TI Assembler and assemble the c99 generated assembler source code for Demo#1. After assembly, select Load and Run option of the E/A and load the following files in this order:

1. load Demo#1 object file
2. place disk with CSUP and load CSUP
3. find the disk with GRF1 and load GRF1

Now, press enter to input entry name: type in START to begin program.

If everything worked well for you an orange ball should be bouncing all around the screen by hitting walls and bumpers. Press any key to get the system "exit-rerun" prompt. Entering yes to the prompt will signal the program to generate new locations for the bumpers. Unfortunately I don't have enough time and newsletter space to explain how the program operates, but if you have any questions please send them to me and I will answer them directly or in the next C Column.

If you find c99 useful, please send your donations to Clint Pulley. He has done a MAGNIFICENT job with the c99 package and should be rewarded for his efforts. Clint has told me that he plans to release a new c99 update sometime in the third quarter of this year. Gifts of this nature will only come with financial support, so please mail those donations. Well, I hope you've learned something from the demos. 'Till next time, 'C' you later...

```
=====
DEMO#1 PROGRAM LISTING:
=====
```

```
/* DEMO #1: "The bouncing Ball"
```

```
Graphic Demo- by Jim Jagielski
```

```
Note: also load grf1 */
```

```
#include dsk1.grf1rf
#include dsk1.random;c
```

```
/* ----- */
```

```
main()
```

```
{
    setup();

    sprite(0,42,7,50,150); /* create ball sprite */
    go();
    text();
}
```

```
/* ----- */
```

```
/* This function setups the graphic display and defines the
   patterns for the characters used. */
```

```
setup()
```

```
{
    int i; /* loop control variable */

    grf1(); /* set graphic mode */
    screen(2); /* set screen color to black */
    color(5,13,13); /* set colors */
    color(6,8,1); /* " " */

    chrdef(48,"182452a9954a2418"); /* bumpers */
    chrdef(42,"3c7ef3fbffff7e3c"); /* ball */

    vchar(1,1,42,72); /* create walls */
    vchar(1,30,42,72);
    hchar(1,1,42,32);
    hchar(24,1,42,32);

    for (i=1; i<=30; i++) /* randomly place bumpers */
        vchar(rnd(20)+3,rnd(24)+5,48,1);
}
```

```
/* ----- */
```

```
/* This function monitors the location of the ball and takes
   the actions needed when the ball hits something. */
```

```
go()
```

```
{
    int rv,cv; /* row and column velocities of ball */
    int rp,cp; /* row and column position of ball */
    int cnt; /* loop control variable used for delay */
```

```

randomize();
spmotn(0,rv=45,cv=40); /* initialize ball motion */
spmot(1); /* set ball in automotion. NOW! */

do /* continuously monitor the position of ball */
{
    spposn(0,&rp,&cp); /* where is ball? */
    if ((rp>174) | (rp<11)) /* hitting top, bottom walls? */
    {
        spmotn(0,rv=-rv,cv);
        for (cnt=0; cnt<100; ++cnt) /* necessary delay */
            poll(0); /* enable interrupts while waiting */
    }
    else if ((cp>225) | (cp<26)) /* hitting left or right walls? */
    {
        spmotn(0,rv,cv=-cv);
        for (cnt=0; cnt<100; cnt++) /* delay */
            poll(0); /* enable interrupts while waiting */
    }
    else if (gchar((rp/8)+1,(cp/8)+1) == 48) /* hitting bumpers */
        spmotn(0,rv,cv=-cv); /* bumpers */
} while (! poll(0)); /* exit loop if key pressed */

spdall(); /* kill all sprites */
}

```

```

=====
DEMO #2 PROGRAM LISTING:
=====

```

```

/* DEMO #2: Simple Inline Push Code Test Program */

```

```

/* NOTE: more than one function is needed to analyze
the time difference between selecting and
NOT selecting Inline Push Code option. */

```

```

main() /* call funct1, funct2 and funct3 32500 times each */
{

```

```

    int i; /* loop control variable */

```

```

    for (i=1; i<=32500; i++)

```

```

    {
        funct1();
        funct2();
        funct3();
    }

```

```

}

```

```

funct1() /* first dummy function */
{

```

```

    ; /* null statement */
}

```

```

funct2() /* second dummy function */
{

```

```

    ; /* null statement */
}

```

```

funct3() /* third dummy function */
{

```

```

    ; /* null statement */
}

```

GRAPHICS PROGRAMMING LANGUAGE
PART 1

Curtis Alan Provance
New Hampshire 99er's User Group

Graphics Programming Language is a compact, byte oriented language developed by TI. GPL is interpreted, meaning that each byte of GPL must be decoded into a series of machine instructions. BASIC's and Extended BASIC's interpreters are written in GPL, as are MULTIPLAN, E/A, and many other cartridges. There are also several console routines written in GPL to which you may LINK. This process is mentioned on page 251 of the Editor Assembler manual.

Unfortunately, information on GPL is very scant. The E/A manual does say that you may link to GPL routines not described on pages 251-257 - provided the routines end with a RETURN byte (>00). Then the manual goes on to other things without ever answering the question - "What other routines?"

The purpose of this and future articles is to clear away the cloud of secrecy which has obscured GPL from many of us. I will be relying heavily on TI99/4A INTERN by Heiner Martin as well as Millers Graphics' excellent EXPLORER program. ROM listings and opcode names are in accordance with those in Mr. Martin's book. I recommend you purchase Mr. Martin's book, though I warn you that you will still spend innumerable hours doing your own decoding. I have done some of the decoding and will share my results with you. Millers Graphics' EXPLORER program is fantastic and a must for the serious hacker. I say up front that any information I put out is "to the best of my knowledge" which means it may be wrong. Please bear with me.

Our explorations of GPL will begin with the GPL interpreter. The heart of the interpreter resides in console ROM from address >0060 to >00B4. The GPL status byte is located at >837C. This byte contains a bit each for High, Greater Than, Condition, Carry, and Overflow (the remaining bits are not used). Remember that the GPL workspace (>83E0) uses registers 13 through 15 as follows: R13 - GROM read data address (>9800); R14 - System flag (>1000); and R15 - VDP write address (>8C02).

DISCUSSION: You should note that most references to VDP and GROM use offsets from these registers, instead of addressing the memory map directly. What I mean is that instead of trying to write a byte from R1 to the GROM address with MOV B R1,>9C02, the interpreter uses MOV B R1,>402(R13) (remember that R13 contains >9800). Why? Because there are routines in the console ROM which will switch GROM addresses in the GPL interpreter. Different addresses would allow you to have several cartridges accessed by the computer within the same program, and have all their titles selectable from the menu - something like a Navarone WIDGET, but without the need for a switch. This would be handy, for example, if you wanted to use the INIT, LOAD, and LINK routines from the E/A cartridge, as well as the 'SPEECH' capabilities of the TE-II module - in the same

BASIC program. Such a device hasn't been commercially available, though I believe Millers Graphics' GRAMKRACKER uses this technique to allow selection of a variety of modules (stored in the GRAMKRACKER) from the main menu.

Here is the core routine with labels representing their address in hex. An explanation follows:

```
L0060      MOV B R6,>402(R13)
          MOV B @>83ED,@>402(R13)
L006A      SZCB @>011B,@>837C
L0070      LIMI 2
L0074      LIMI 0
          MOV B #R13,9
          JLT L0086
          MOV B R9,R4
          SRL R4,12
          MOV @>0C36(R4),R5
          B #R5
L0086      CLR R4
          MOV R9,R5
          ANDI R5,>0100
          BL @>077A
          SWPB R4
          MOV R1,R3
          MOV R0,R2
          CI R9,>A000
          JL L00B0
          CDC @>0030,R9
          JNE L00BC
          MOV R13,R1
          MOV B #R1,R0
          DEC R1
          BL @>07AA
          JMP L00C0
L00B0      MOV R9,R8
          SRL R8,8
          SETO R0
          MOV @>0BFE(R8),R8
          B #R8
L00BC      BL @>077A
L00C0      MOV R9,R8
          SRL R8,9
          MOV @>0C4E(R8),R8
          C R2,R0
          B #R8
```

L0060: Load register 6 with the desired GPL address and branch here. The address will be loaded into the GROM

address (>83ED is R6's lower byte). This is used by the various GPL branch instructions: Branch, Branch if Reset, and Branch if Set. Being Set or Reset refers to the Condition bit of the GPL status byte at >837C. The various branch bytes will be discussed later.

L006A: This is used to clear the condition bit of the GPL status byte - useful when you want to use a Branch on Reset as your next command (two bytes) rather than a Branch (three bytes). It's also used merely to clear the status byte before continuing.

L0070: This is the normal return to the interpreter. Neither the GROM address nor the status byte is changed. Interrupts are enabled to service peripherals, move sprites, count off sound durations, check for the QUIT key, etc.

L0074: Some routines may return here instead of L0070 if they don't want an interrupt serviced; interrupts may alter the current VDP or GROM address.

The next byte from GROM is moved into the high byte of R9. If it is >00 to >7F, it is handled by a branch table located at >0C36. If the byte is >80 to >FF, it is handled by a routine at L0086. The branch table at >0C36 is based on the first nybble of the byte - with the last bit ignored. For example, if the original byte from GROM were >03 or >13, the branch address would come from >0C36(>00) - remember, the last bit of the first nybble is ignored, so 1 looks like 0. The four branch addresses and the bytes which call them are as follows (X means we don't care what the nybble is):

BYTES:	ADDRESS:	FUNCTION:
>0X, >1X	>0270	ASSORTED
>2X, >3X	>061E	MOVE BYTES
>4X, >5X	>011A	BRANCH ON RESET
>6X, >7X	>010E	BRANCH ON SET

The 'ASSORTED' functions have a further branch table which we will discuss shortly. The MOVE instructions are further decoded, along with additional bytes, to determine the number of bytes to move, what the source is, and what the destination is. Sources can be GROM, VDP, and CPU. Destinations can be VDP, CPU, VDP registers and - in certain circumstances - GRAM. More on MOVE's later. The Branch on Reset and Branch on Set are simple to describe:

```
L010E
    MOV B @>837C,R4
    SLA R4,2
    JLT L0122

L0116
    MOV B *R13,4
    JMP L006A

L011A
    MOV B @>837C,R4
    SLA R4,2
    JLT L0116

L0122
```

```
MOV B *R13,@>83F3
ANDI R9,>1FFF
MOV B @>0002(R13),R6
ANDI R6,>E000
SOC R9,R6
JMP >0060
```

Think of these more as jump instructions, with the base address being the beginning of the GROM chip in use. Additionally, both the Branch on Reset (BR) and Branch on Set (BS) require another byte from GROM. Glue the two bytes together, mask off the first three bits, and you have the size of the 'jump'. For example, suppose you were in GROM 1, which starts at >2000. The interpreter reads in a byte of >74. It decodes this as a Branch on Set and branches to L010E. First, the GPL status byte is copied into R4. Next, the condition bit is checked. If it is set, the branch is executed (at L0122) otherwise, the rest of the address is read into R4 (and discarded) and the interpreter continues. Assume the condition bit was set. The rest of the 'jump' is read into the low byte of R9 (>83F3) - the second byte happens to be >38. The whole word is now >7438 and is stored in register 9. Mask off the first three bits of register 9 to get >1438. Now make a copy of the first byte of the current GROM address in R6. Again, notice we are using @>0002(R13) instead of >9802. Assume the current address is >2546. Mask off all but the first three bits of R6 to get >2000. Finally, for each bit in R9 set the corresponding bit in R6. The interpreter now has the value of >3438 in register 6. Jump to >0060 to load the new GPL address in R13 and you have performed a GPL branch! Branch on Reset is similar.

L0086: This is the interpreter for negative bytes. R4 is cleared because it will be used later to hold various 'flags'. A copy of the GPL byte (R9) is made in R5 then masked off except for bit 7. This bit flags routines that the address is made up of two bytes instead of one. The routine at >077A is invoked to get the destination address. We'll look at this later (it's rather complicated). I should mention, though, that the routine at >077A was written to get the source address and/or byte in R1 and R0. Since we are getting the destination first, we move the values into R3 and R2 and move the 'flag' value in R4 from the low byte (source flags) to the high byte (destination flags). Now look at the GPL byte itself. If it is >8X or >9X, then it is ready to be executed and we jump to the routine at L00B0. Otherwise, we need to check to see if the 'source' is an immediate value. The value at >30 is >0200 (LIMI instruction) so we are checking the sixth bit. If it is set, the source is immediate (in GROM). Otherwise, the source must be retrieved from memory somewhere - GROM, VDP, or CPU. Regardless of the source and destination values, everything eventually comes down to another branch table. The bytes >8X and >9X use a branch table located at >0BFE while bytes >AX through >FX use a branch table located at >0C4E. We'll look at each of these later.

****NOTE:** FREEWARE means you are **expected** to pay the author of the disk what you feel the disk is worth, usually <\$10. We only charge for the blank disk and copying. Make FREEWARE work! **PAY!**

DISKNAME	LANGUAGE	COMMENTS	COMMENTS	COMMENTS
ADRSMaster	* XBASIC	JHB database program.	LINKMASTER&DVECTOR needed	*12
ADVENTURE1	BASIC	text and graphic adventure games		
ASGARD/PD1	XBASIC	Public domain music software from Asgard.		
ASGARD/PD2	XBASIC	Public domain music software from Asgard.		
ASGARD/PD3	XBASIC	Public domain music software from Asgard.		
ASGARD/PD4	XBASIC	Public domain music software from Asgard.		
ASGARD/PD5	XBASIC	Public domain music software from Asgard.		
ASSY GAMES	XBASIC	arcade quality games. 32K/speech synth. required		
B TRAVEL#1	XBASIC	good demos and utilities on this FREEWARE disk.		
BA-WRITER	* XBASIC	Best FREEWARE TI-WRITER package! A must!	> \$5 FLIPPY	
BA-WRI-DOC	* TI-WTR	DIS/VAR80 files for the above	/	
BASICS1-9	BASIC	T.I.'s basic lessons on disk.		
BBS	* XBASIC	John Clulow's bulletin board software	FREEWARE disk.	*5
BEST/SONGS	XBASIC	Bill Knecht's tunes with graphics.		
BEST/HYMNS	XBASIC	Bill Knecht's hymns with graphics.		
BLUEY	XBASIC	create and animate your sprites with this utility.		
C99REL1	* E/A	Curt Pulley's FREEWARE "C" language.	\	
C99UPDATE	* E/A	Modifications to the C99REL1 disk.	> \$6.50	
C-TUTORIAL	* E/A	Demo programs and info for "C".	/	
CALENDAR	XBASIC	FREEWARE disk of calendar programs with documentation.		
CALENDAR2	XBASIC	makes a calendar with notation.		
CARTBUSTER	E/A--BK	save rom/prom cartridges to disk.	SUPERCARTRIDGE required	
CASHFLOW	XBASIC	financial programs.		
CONGOBONGO	E/A	arcade quality game.		
COPY/CATXB	XBASIC	John Clulow's disk copier program.		
CUBIT	XBASIC	arcade quality game.		
DASSM*V1/3	E/A	disassemble your assembly programs.		
DAYTONA99	XBASIC	collection of demos/utilities from Daytona User Group.		
DAVIDDISK	XBASIC	programs from Davis' book.		
DEBUGGER	E/A	debug your assembly language programs.		
DIAGNOSTIC	XBASIC	T.I.'s test disk for the 99/4A.		
DISKE	E/A	Sector access program. inspect and/or modify sectors.		
DM10003/1	-----	DM II replacement. Loads from E/A, TI-WRITER, XBASIC.		
-DIRECTOR-	XBASIC	Gives good, fast, sorted directory for your library.		
DVECTOR	XBASIC	JHB database program. ADRSMaster and LINKMASTER needed.		
DVUG/2D5	XBASIC	Shuttle-graphics and music from Delaware UG.		
EE-LIBRARY	* BASIC	T.I.'s idea of electrical engineering programs.		*5
FAST-TERM	E/A	Terminal emulator program that supports TE II and XMODEM.		
FINANCE	XBASIC	Financial programs.		
FUNLWRITER	XBASIC	Good XBASIC loader version of TI-WRITER.		
GAMES 01	XBASIC	4 games. CHINACHESS is interesting.		
GEMINI	XBASIC	A disk full of 10-X demos.		
GLPDEMO	XBASIC	Printer demos for the Centronics GLP printer.		
GRADEBOOK	XBASIC	Gradebook and flashspelling programs w/documentation.		
GRAPH-PACK	* BASIC	T.I.'s idea of graphing programs.		*6
GRAPHX	GRAPHX	6 pictures to be used with the GRAPHX package.		
HBMPRINT	HBM	Dump HBM files to printers.		
INCOME TAX	MP	1984 income tax template for multiplan.		
ISAM	XBASIC	ISAM files as described in MICROpendium 12/84 page 35.		
ISS	XBASIC	Arcade games.		
ISS/MUSIC	XBASIC	Music programs.		

JET-DSK01A * XBASIC Good FREEMWARE disk from John Taylor. \
 JET-DSK01B * XBASIC More of the above. > *5 FLIPPY
 JET-DSK02A * XBASIC John Tavlör FREEMWARE sprite building program.> *5 FLIPPY
 JET-DSK02B * XBASIC 127 sprites for the above FREEMWARE disk. /
 KNIGHTDISK XBASIC FREEMWARE disk of utilities from Knight (TK-WRITER).
 LINKMASTER XBASIC JHB database program. DVECTOR and ADRSMASIER needed.
 LOGO_DISK LOGOII Good demo of LOGOs power.
 MASSCOPY XBASIC Latest version of this FREEMWARE disk copier. V3.25
 MASTERDISK XBASIC Directory program.
 MEGABUCKS XBASIC Megabucks number selection program.
 MENTOR XBASIC Barqraph, monopoly, draw-poker, other goodies.
 MTXT/DISK MINIMEM Minimemory utilities for 40 column.
 MUSIC XBASIC Moore music programs for the T.I.
 MUSIC2 XBASIC Moore music programs for the T.I.
 MUSICOMPLR E/A BASIC loader allows music playing as other program runs.
 MUSIC MAKR MUSIC Music for the MUSICMAKER cartridge.
 NEATLIST XBASIC FREEMWARE programming aid disk from Danny Michael.
 NH99ERS#1 XBASIC Games, music, word processor, speech, and graphics.
 NH99ERS#2 XBASIC Games for the T.I.
 NH99ERS#3 XBASIC Graphics, games, and utilities.
 99WRITERII XBASIC Another TI-WRITER.
 ON_DISKJ/A ----- Programs from HCM in various languages.
 ON_DISK4_1 ----- Programs from HCM in several languages.
 ON_DISK4_4 ----- Programs from HCM in several languages.
 ON_DISK4_5 ----- Programs from HCM in several languages.
 ON_DISK5_1 ----- Programs from HCM in several languages.
 ON_DISK5_2 ----- Programs from HCM in several languages.
 ON_DISK5_3 ----- Programs from HCM in several languages.
 ON_DISK5_4 ----- Programs from HCM in several languages.
 ON_DISK5_6 ----- Programs from HCM in several languages.
 OSCAR1 BASIC Programs from the OSCAR reader.
 OSCAR2 BASIC Programs from the OSCAR reader.
 PILOT * E/A The PILOT language. \FLIPPY
 PILOT DOC * XBASIC PILOT documentation w/print function. (1.5hrs !) > *5
 POTPOURRI XBASIC Games and utilities. Includes diskjacket/disk labeler.
 PRBASE * XBASIC Database. \
 PRBASE/DOC * XBASIC Instructions for above. > *5 FLIPPY
 PROGAID123 BASIC T.I.'s programming aids in both basic and xbasic.
 RAM/SOFT XBASIC Craps game.
 RODSK200 XBASIC Gamma match antenna design for hams.
 SAMUSIC/1 XBASIC Moore songs.
 SAME/DIFF XBASIC Kids matching games. Speech required.
 SAMSGAMES1 * XBASIC Programs from the SAMS book. \
 SAMSGAMES2 * XBASIC Continuation of above. > *5 FLIPPY
 SCREENDUMP XBASIC Or basic. A must FREEMWARE disk from Danny Michael.
 SILVERWOLF E/A Assembly language utilities.
 >>SPACE!<< XBASIC Space games with graphics. not as flashy as some.
 SPCHTRADE XBASIC See and hear Lincoln speak! Speech synthesizer required.
 SUPERBUGII E/A Debugger. Including one for the E/A-BK SUPERCARTRIDGE.
 TAX-INVEST TIMP Multiplan overlay.
 TE3-DIALER E/A Good terminal emulator for ASCII files.
 TI/DEMO BASIC Several demos T.I. gave to their dealers.
 TIPS XBASIC Tips from the Tigercub.
 TI-SINGS TE II Hear your computer sing! Speech synth. required.
 TI-SINGSXB XBASIC Xbasic version of above. NH99ER exclusive.
 TI-TIPS_01 * TIWTR DIS/VAR00 files with useful tips. GOOD reading. > *5
 TI-TIPS_02 * TIWTR More of the above. /


```

TI-TIPS_03 * TIWTR More of the above. \
TI-TIPS_04 * TIWTR More of the above. > $5 FLIPPY
TI-SORT E/A-BK Assembly language sort routines for E/A-BK SUPERCARTRIDGE.
TIMP&TIWRT WTR-MP Updates for TI-WRITER and MULTIPLAN.
TIWRITER'S TI-WTR Reference material for FUNLWRITER and others.
TIWRTPIOUP TIWTR PIO version of TI-WRITER.
TK*S*DUP XBASIC Disk copier program.
TRIVIA99ER XBASIC Trivia database.
TRIVIABASE XBASIC Another trivia database.
VIDEODEMOS XBASIC A must have graphics demo.
WORDCOUNT XBASIC FREWARE assy. lang. utility. GOOD
WORKHORSE XBASIC Good collection of utilities. Don't miss this one!
XB-GAMES XBASIC Arcade quality games.
XB-GAMING1 XBASIC More arcade quality games.
XB-LESSONS XBASIC T.I.'s xbasic lessons on disk.
XB-WRITER1 XBASIC One of the best TI-WRITER disks available.
XBASIC-UT1 * XBASIC Great utility disk from Travers. > $5 FLIPPY
XBASIC-UT2 * XBASIC More of the above. /
X_D E/A Assembly language utility and demos.

```

F O R T H D I S K S

****NOTE:** Many 4TH disks have program information on screen 2 and screen 3. Use LIST or -PRINT to see these screens. Many of the "programs" on these disks are well documented.

DISKNAME	LANGUAGE	COMMENTS	COMMENTS	COMMENTS
4TH/BACKUP	XBASIC	Disk copier program in FORTH.	Donation requested.	
4TH_CLONER	E/A	Another disk copier program in FQTH.		
4thDOODLES	E/A	Good bitmap graphics demo.		
DATADISK02	E/A	Demos, utilities, and games.	read SCRs 1, 2, and 3.	
DATADISK03	E/A	More of the above.		
DATADISK04	E/A	More of the above.	Includes the game COSMIC CONQUEST.	
SOURCE-A	E/A	The sorce code for T.I. FORTH.		
SOURCE-B	E/A	Part 2 of the sorce code for T.I. FORTH.		
SYS-DISK02	E/A	Modified 4TH disk with autorepeat, etc..		
TE4TH	E/A	Terminal emulator program in FORTH.		
TI-FORTH	E/A	T.I.'s original FORTH disk.		
UTILITY4TH	E/A	Manv utilities plus HELP and instruction screens.		
UTILSOURCE	E/A	Sorce code for UTILITY4TH.		
VOLKFORTH1	E/A	More 4TH goodies. Check line 0 of all SCRs for clues.		
VOLKFORTH2	E/A	Mote of the above.		
XB-FORTH	XBASIC	4TH version that loads from XBASIC.		
FORTHXLD1	XBASIC	Two disk modified 4TH disk set.		
FORTHXLD2	XBASIC	continuation of above.		


The service charge per DISKNAME is \$3.50 unless otherwise noted. The asterisk after a DISKNAME indicates that the increased charge is due to printed documentation. PLEASE... it is impossible to anticipate everyone's needs. Even if you are only interested in a selection, let the librarian know so he can have copies ready for the meeting. **NO COPYING WILL BE DONE AT THE MEETINGS!** As you know, our disks are supplied with labels and are write-protected. Please use these disks as your **MASTERS** only! From time to time there have been updates to programs such as DM1000, which we provide free of charge for those who have "bought" the software from us. Your MASTER must be returned for proper credit. Thank you for your patience in awaiting this listing. If you have further questions, please contact the librarian: Richard J. Bailey - 68A Church Street - Gonic, NH 03867 (603) 332-7855

MAIL ORDERS: Please add an additional \$1 S/H charge per order!!

BONANZA
Next Time Make It

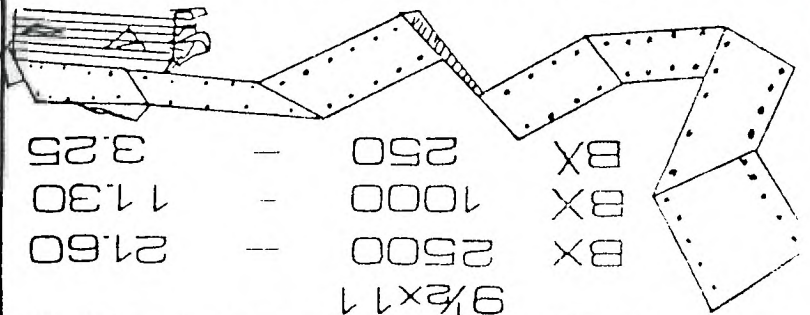
DECEMBER 7
NOVEMBER 2
OCTOBER 5
SEPTEMBER 7
AUGUST 4
JULY 6
JUNE 1
MAY 4

SCHEDULED MEETINGS
FOR 1987:



ORIS
co. (603) 668-4245
456 Beech St. Manchester, NH

Your Computer Forms & Supplies Headquarters
ALL SIZES AVAILABLE — MOST IN STOCK



BX	250	—	3.25
BX	1000	—	11.30
BX	2500	—	21.60

9 1/2 x 11
20* BLANK EASI-PERF

— Free Local Delivery —

COMPUTER PAPER

NEW HAMPSHIRE 99'ERS USER GROUP, INC.
PO BOX 5991
MANCHESTER, NH 03108-5991



**PLEASE SUPPORT BONANZA
— THEY SUPPORT US!**